**AFRL-IF-RS-TR-2005-258**
**Final Technical Report**
**July 2005**


# HIGH PERFORMANCE COMPUTING (HPC) CHALLENGE (HPCC) BENCHMARK SUITE DEVELOPMENT

**CACI Technologies, Incorporated**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. Q455**



**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

**STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-258 has been reviewed and is approved for publication

APPROVED: /s/

CHRISTOPHER FLYNN
Project Engineer

FOR THE DIRECTOR: /s/

JAMES A. COLLINS, Acting Chief
Advanced Computing Division
Information Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | JULY 2005 | Final Sep 03 – Oct 04 |

**4. TITLE AND SUBTITLE**
HIGH PERFORMANCE COMPUTING (HPC) CHALLENGE (HPCC) BENCHMARK SUITE DEVELOPMENT

**5. FUNDING NUMBERS**
C - F30602-00-D-0021
PE - 62301E
PR - Q455
TA - QF
WU - 26

**6. AUTHOR(S)**
Jack Dongarra

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Prime:
CACI Technologies, Incorporated
1300 Floyd Avenue
Rome New York 13440

Sub:
University of Tennessee
1122 Volunteer Boulevard
Knoxville Tennessee 37996-3450

**8. PERFORMING ORGANIZATION REPORT NUMBER**
N/A

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTC
26 Electronic Parkway
Rome New York 13441-4505

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
AFRL-IF-RS-TR-2005-258

**11. SUPPLEMENTARY NOTES**

AFRL Project Engineer: Christopher Flynn/IFTC/(315) 330-3249/ Christopher.Flynn@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** ***(Maximum 200 Words)***
The goal of performance modeling is to measure predict, and understand the performance of a computer program or set of programs on a computer system. The applications of performance modeling are numerous, including evaluation of algorithms, optimization of code implementation, parallel library development, and comparison of system architectures, parallel system design, and procurement of new systems. The overall objective of this effort was to survey a number of DoD related applications in an effort to ascertain their needs with respect to determining what metrics exist, what metrics need to be developed.

**14. SUBJECT TERMS**
High Performance Computing, Benchmarks, Metrics

**15. NUMBER OF PAGES**
22

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

# Table of Contents

## Summary

The work for this project was comprised of the following major areas:

- Provide a focused research and development program, creating new generations of high-end programming benchmarks in order to realize a new vision of high-end computing, High Productivity Computing Systems (HPCS).
- Expose the issues of low efficiency, scalability, software tools and environments, and growing physical constraints.
- Characterize architecture performance of parallel systems being developed for the DAPRA High Productivity Computing Systems Program.
- Develop software for the benchmarking and performance evaluation of key components of high performance systems.
- Develop methods for guiding the collection of performance data and for analyzing and abstracting from measured performance data.
- Help promote this effort in the community.

The objectives of this effort were:

- To establish a comprehensive set of parallel benchmarks that is generally accepted by both users and vendors of parallel systems.
- To provide a focus for parallel benchmark activities and avoid unnecessary duplication of effort and proliferation of benchmarks.
- To set standards for benchmarking methodology and result-reporting together with a control database/repository for both the benchmarks and the results.
- To make the benchmarks and results freely available to the public domain.
- To engage the high-performance community in helping define the future expansion of the benchmark collection.
- To run HPC Challenge over a range of parameters.
- To collect and make available performance results in a standard web based format.
- To compute software and hardware metrics.
- To apply the run time tools being studied by the Execution Time modeling group to HPC Challenge.

## Introduction

Unfortunately, much of the literature focuses on ad hoc approaches to the evaluation of systems rather than on the potential standardization of the benchmark process. If benchmarking is to mature sufficiently to meet the requirements of system architects as well as application and algorithm developers, it must address the issue of standardization.

A number of projects such as Perfect, NPB, ParkBench, and others have laid the groundwork for what we hope will be a new era in benchmarking and evaluating the performance of computers. The complexity of these machines requires a new level of detail in the measurement and comprehension of the results. The quotation of a single number for any given advanced architecture is a disservice to manufacturers and users alike, for several reasons. First, there is a great variation in performance from one computation to another on a given machine; typically the variation may be one or two orders of magnitude, depending on the type of machine. Secondly, the ranking of similar machines often changes as one goes from one application to another. So, for example, the best machine for circuit simulation may not be the best machine for computational fluid dynamics. Finally, the performance depends greatly on a combination of compiler characteristics and the human effort that was expended on obtaining the results.

## Methods, Assumptions, and Procedures

This first phase of the project developed, hardened, and reported on a number of benchmarks. The collection of tests included tests on a single processor (local) and tests over the complete system (global). Each examined performance evaluation for spatial locality and temporal locality. The tests on a local basis include DGEMM, STREAM, RandomAccess, and FFT and the tests on a global basis included High Performance Linpack, PTRANS, RandomAccess, and FFT.

The most reliable technique for determining the performance of a program on a computer system is to run and time the program (multiple times), but this can be very expensive and it rarely leads to any deep understanding of the performance issues. It also does not provide information on how the performance will change under different circumstances (e.g., scaling the problem or system parameters, or porting to a different machine).

An alternative approach to running the actual application codes is to develop a set of representative benchmark programs and to run these benchmarks on various systems with various problem and system sizes. The problem with this approach is that a quantitative analysis of the measured data is necessary to allow a deeper understanding and interpretation - i.e., abstraction of the measured results. Statistical analysis techniques require a large amount of data to be collected. However, collecting data for all possible system and problem parameter settings is impractical. Hence, a determination needs to be made of what and how much data needs to be collected to provide an adequate basis for sound analysis.

Another approach is to generate a model of the program and the computer system and use the model to make performance predictions, varying model parameters to simulate varying program and computer system parameters. The difficulty with this approach is in generating and validating the model. The performance of production-level application codes is a result of complex interactions between processor architecture, memory access patterns, the memory hierarchy, the communication subsystem, and the system software. Modeling each of the complex components of the system alone is a challenge. Still more challenging is the task of accurately modeling the interactions between components and the performance of complex application codes on the entire system.

Our approach in the second phase of this effort was to investigate the performance modeling problem by combining benchmarking, statistical analysis, and hierarchical modeling techniques to produce accurate models that can predict the performance of complex applications on today's and tomorrow's high performance systems.

Although this work did not directly address performance engineering of complex application codes, our work laid the basis for the construction of parallel libraries that allow the reconstruction of application codes on several distinct architectures so as to assure performance portability. Once the requirements of applications are well understood, one can construct a library in a layered fashion.

The overall objective of this effort was to survey a number of DARPA related applications in an effort to ascertain their needs with respect to determining what metrics exist and what metrics need to be developed. In the course of this effort we helped in defining the metrics for future productivity, in particular:

- Temporal data locality measures the memory access patterns' reuse of data in CPU time domain. In other words, it measures the likelihood of a datum to be used in two close points in time.
- Spatial data locality measures the memory access patterns' reuse of data in memory address space domain. In other words, it measures the likelihood of two data being used provided that they are close to each other in memory.
- Source Line Of Code (SLOC) count is a simple yet very effective measure of code complexity and in turn characterizes very well the human effort involved in writing, maintaining, and refactoring a piece of code.

Using the above metrics, a set of representative application kernels were selected that reveal system performance and productivity under the workloads of varying values of the metrics so that bounds can be established for end-user applications.

## Integrity of the Benchmark Code

The HPCC benchmark includes existing and well known benchmark codes as well as not so well known codes that were not intended for benchmarking by their authors. In both cases one may argue that it is possible to obtain HPCC-equivalent functionality by running each of the included tests separately or some subset thereof. Based on our extensive experience in high performance benchmarking, utilization of the entire suite, up to this point, offers as complete an analysis of benchmarking as has ever been done. There are a few important reasons why performing individual or subset tests would be neither practical nor complete:

***Uniform verification***

Each code was examined and (as necessary) augmented with a robust verification procedure that ensures numerical correctness of the result. This is in sharp contrast to traditional forms of benchmarking that only focus on best performance

(or best time).

***Reasonable optimization***

In the hands of a skillful benchmarking engineer, each of HPCC's individual tests alone can be optimized beyond any skillful user's comprehension. Such a scenario is made unlikely with an HPCC framework that encapsulates all the tests in a single runtime thus excluding the possibility of switching the tested system into a special mode that would only benefit a single test. Another contribution of HPCC is the transfer of knowledge from the benchmarking engineer to the user as the optimization techniques are meant to be disclosed by the party submitting results.

***Convenience and correctness***

Running each of HPCC's tests separately requires manual effort, which is cumbersome, costly, and error prone if done in a robust and reliable manner. HPCC eliminates this by automating the process of gathering performance data on widely applicable hardware characteristics.

## Results Overview

As a result of this project, software was developed in an open source mechanism and distributed to the community via the normal channels for open source software – a publicly available web page is used for downloading stable releases of the software while read-only CVS access may be used for development snapshots. Occasionally, the source code was distributed via email if other means were inaccessible. There was a monthly phone call and/or Access Grid meetings of participants to exchange ideas and progress as well as an electronic mailing list to assist with participant communication. Finally, the lead participants participated in face-to-face meetings as needed, and there has been at least one such meeting annually.

The problem area may be characterized by most common memory access patterns and is defined by seven benchmarks: HPL, DGEMM, STREAM, PTRANS, RandomAccess, FFT, and Latency/Bandwidth:

- HPL is the Linpack Toward Peak Performance (TPP) benchmark. The test

stresses the floating point performance of a system.

- DGEMM measures the floating point rate of execution of double precision real matrix-matrix multiplication.
- STREAM is a benchmark that measures sustainable memory bandwidth (in GB/s) of simple vector application kernels.
- PTRANS (from the ParkBench suite) measures the rate of transfer for large arrays of data between multiprocessor's memories.
- RandomAccess measures the rate of integer updates of random memory locations.
- FFT measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).
- Latency/Bandwidth measures (as the name suggests) latency and bandwidth of communication patterns of increasing complexity between as many nodes as is time-wise feasible.

## Benchmark Data

To ensure broad impact and scientific value of the benchmark suite, results from a vast array of data are collected for each submission. The data is gathered in a database that has read-only public access. The data in the database can be divided into the following categories:

- CPU parameters such as floating-point execution rate (measured in Gflop/s) of various computational kernels.
- Memory subsystem parameters such as data transfer rates (measured in GB/s) for various CPU workloads and memory bus sharing scenarios.
- Communication subsystem parameters such as transfer rates (measured in GB/s) across the system interconnect and message latencies (measured in micro-seconds).
- Hardware, software, and productivity optimizations including complete descriptions of the hardware configuration, software environment and tools that were used to produce the executable, and all the specific changes applied to the optimized run.

## Benchmark Optimization and Result Database

An integral part of HPCC is the database that stores various optimization techniques applicable to the HPCC tests as well as the results of applying these optimizations. All of the data is time stamped and consistently gathered from every system submitted to the HPCC website. As such, it constitutes an invaluable resource for hardware and software vendors as well as application vendors.

The optimization portion of the database stores two types of information:

1. Hardware/software optimization

   This portion includes reports on how the programming environment and system libraries influence hardware and its performance.

2. Productivity optimization

   This portion of the database shows how the human factor is included in the overall system design. In particular the result submitters describe changes made for the optimized run of the benchmark. Based on this information, conclusions may be drawn about feasibility and actual performance gains for end-applications.

The results portion of the database includes various computer system parameters gathered during benchmark run. In particular, the data may be divided into three groups:

1. Processor parameters

   These include floating-point execution rates for computational kernels such as global linear equation solving, local matrix multiplication, and local/global FFT. Numerical capabilities of the processor are also noted by measuring relevant numerical norms that assess correctness and quality of the delivered solution.

2. Memory parameters

   These include transfer rates of multiple sorts that show performance of the

simplest data movement scenarios (CPU-memory transfer) and more elaborate schemes that involve multiple CPU calculations combined with simultaneous accesses to multiple memory modules.

3. Interconnect parameters

   Essentially, two types of parameters are considered: latency and bandwidth. The measurement scenarios used to obtain them vary greatly from the simple polling-driven scheme with only two interconnect end-points exchanging data in a synchronous fashion to network-capacity, limited tests of raw communication and computation-interleaved probes that rely on communication system throughput and tolerance of high volume traffic.

## Rules for Running the Benchmark

The HPCC rules ensure integrity of the benchmark code when run repeatedly on the user system by requiring a wide range of data to provide exhaustive information about the system used for benchmarking and conditions under which the run took place. Due to the fact that the HPCC benchmark is already being used in procurement cycles at many supercomputing centers, we maintain the rules and make them available at the web site. The rules are included here verbatim:

There must be one baseline run submitted for each computer system entered in the archive. There may also exist an optimized run for each computer system.

1. Baseline Runs: Optimizations as described below are allowed.
   - Compile and load options

     Compiler or loader flags, which are supported and documented by the supplier, are allowed. These include porting, optimization, and preprocessor invocation.
   - Libraries

     Linking to optimized versions of the following libraries is allowed:
     - BLAS
     - MPI

     Acceptable use of such libraries is subject to the following rules:

- All libraries used shall be disclosed with the results submission. Each library shall be identified by library name, revision, and source (supplier). Libraries, which are not generally available, are not permitted unless they are made available by the reporting organization within six months.
- Calls to library subroutines should have equivalent functionality to that in the released benchmark code. Code modifications to accommodate various library call formats are not allowed.
- Only complete benchmark output may be submitted - partial results will not be accepted.

2. Optimized Runs
   - Code modification

     Provided that the input and output specification is preserved, the following routines may be substituted:
     - In HPL: HPL_pdgesv(), HPL_pdtrsv() (factorization and substitution functions)
     - no changes are allowed in the DGEMM component
     - In PTRANS: pdtrans()
     - In STREAM: tuned_STREAM_Copy(), tuned_STREAM_Scale(), tuned_STREAM_Add(), tuned_STREAM_Triad()
     - In RandomAccess: MPIRandomAccessUpdate() and RandomAccessUpdate()
     - In FFT: fftw_malloc(), fftw_free(), fftw_create_plan(), fftw_one(), fftw_destroy_plan(), fftw_mpi_create_plan(), fftw_mpi_local_sizes(), fftw_mpi(), fftw_mpi_destroy_plan() (all these functions are compatible with FFTW 2.1.5 so the benchmark code can be directly linked against FFTW 2.1.5 by only adding proper compiler and linker flags, e.g. -DUSING_FFTW).
     - In Latency/Bandwidth component alternative MPI routines might be used for communication. But only standard MPI calls are to be preformed and only to the MPI library that is widely available on the tested system.

- Limitations of Optimization
  - Code with limited calculation accuracy

    The calculation should be carried out in full precision (64-bit or the equivalent). However the substitution of algorithms is allowed (see Exchange of the used mathematical algorithm).
  - Exchange of the used mathematical algorithm

    Any change of algorithms must be fully disclosed and is subject to review by the HPC Challenge Committee. Passing the verification test is a necessary condition for such an approval. The substituted algorithm must be as robust as the baseline algorithm. For the matrix multiply in the HPL benchmark, Strassen Algorithm may not be used as it changes the operation count of the algorithm.
  - Using the knowledge of the solution

    Any modification of the code or input data sets, which uses the knowledge of the solution or of the verification test, is not permitted.
  - Code to circumvent the actual computation

    Any modification of the code to circumvent the actual computation is not permitted.

## Project's Website

In order to provide easy access to the results of this project a publicly available web site was developed. The website can be accessed at http://icl.cs.utk.edu/hpcc/ (the site has nearly 2000 visitors per month and has been used to download the benchmark code by almost 1000 visitors). It consists of the following components:

- Rules for running the benchmark and reporting results.
- News items from external media outlets.
- Download page that allows downloading of the benchmark code in various forms and versions.
- Frequently Asked Questions has an extensive list of questions frequently encountered in benchmarking and pertaining to the HPCC Suite.
- Resource page with (mostly external) links related to the benchmark suite.

- Pages with sponsors and collaborators that made the project possible.
- Web form for submitting information about tested system and the output of the benchmark.
- Database read-only interface that allows users to interactively obtain various views of the submitted data or to export the contents of the database for archiving or for a more thorough analysis on the user system.

## Conclusion

The impact of this work on the community is the availability of an easy mechanism to test, evaluate, and compare high productivity systems. The applications of performance modeling are numerous, including evaluation of algorithms, optimization of code implementations, parallel library development, comparison of system architectures, parallel system design, and procurement of new systems.

The main components of the HPC Challenge Benchmark Suite are based on existing codes that are well known and used in the HPC community. This fact greatly contributes to wider adoption of our effort. In addition, our framework combines these existing codes together in a unique way by defining very specific rules about the conditions under which they should be run and some of the included tests are used in new scenarios. We also largely expanded on the benchmark deployment and results data to facilitate fairness of performance assessment.

## References


- Alice E. Koniges, Rolf Rabenseifner and Karl Solchenbach: “Benchmark Design for Characterization of Balanced High-Performance Architectures,” In IEEE Computer Society Press, www.computer.org/proceedings/, *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, Workshop on Massively Parallel Processing (WMPP), April 23-27, 2001, San Francisco, USA.
- Rolf Rabenseifner and Alice E. Koniges: “Effective Communication and File-I/O Bandwidth Benchmarks,” In J. Dongarra and Yiannis Cotronis (Eds.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, *Proceedings of the 8th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2001*, Santorini, Greece,

Sep. 23-26, LNCS 2131, pp 24-35.

- Jack Dongarra, Piotr Luszczek, and Antoine Petitet: "The LINPACK Benchmark: Past, Present, and Future," *Concurrency and Computation: Practice and Experience* 15(9):803-820, August 2003, ISSN 1532-0634.
- John D. McCalpin, STREAM: Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/
- Antoine Petitet, R. Clint Whaley, Jack Dongarra, Andy Cleary, HPL - A Portable Implementation of the High Performance Linpack Benchmark for Distributed-Memory Computers, http://www.netlib.org/benchmark/hpl/
- PARKBENCH - PARallel Kernels and BENCHmarks, http://www.netlib.org/parkbench/
- Jack Dongarra and Piotr Luszczek: "Introduction to the HPC Challenge Benchmark Suite," *Computer Science Department Technical Report 2005*, UT-CS-05-544, University of Tennessee Knoxville.
- Stephen Shankland. "Supercomputer ranking method faces revision", Published June 18, 2004. CNET News.com, http://news.com.com/Supercomputer+ranking+method+faces+revision/2100-7337_3-5238405.html?tag=nefd.lede.
- Jan Stafford. "Cray CTO: Supercomputers outshine Linux clusters in HPC, Part 1", Published August 24, 2004. SearchEnterpriseLinux.com, http://searchenterpriselinux.techtarget.com/originalContent/0,289142,sid39_gci1000889,00.html
- Mark Hachman. "Alternative Supercomputer Metrics Sought", Published November 8, 2004. eWEEK.com, http://www.eweek.com/article2/0,1759,1720493,00.asp.
- Christopher Lazou. "HPC Benchmarks: Going for Gold in a Computer Olympiad?", Publised January 21, 2005. HPCwire, http://www.tgc.com/hpcwire/hpcwireWWW/05/0121/109098.html
- "Cray XD1 Supercomputer Outscores Competition in HPC Challenge Benchmark Tests; Benchmarks Assess Real-World Performance in Several Key Aspects of Supercomputing", published February 14, 2005. BusinessWire http://home.businesswire.com/portal/site/google/index.jsp?ndmViewId=news_view&newsId=20050214005533&newsLang=en

# Appendix

***Benchmark Code***

The initial public release of the benchmark included over 40,000 lines of C code. A compact reference implementation is included in this report that gives a flavor of the actual implementation. The reference implementation is written in a popular scripting language called Python.

- HPL – Linpack reference implementation

```
import numarray, time
import numarray.random_array as naRA
import numarray.linear_algebra as naLA
n = 1000
a = naRA.random([n, n])
b = naRA.random([n, 1])
t = -time.time()
x = naLA.solve_linear_equations(a, b)
t += time.time()
r = numarray.dot(a, x) - b
r_n = numarray.maximum.reduce(abs(r))
print t, 2.0e-9 / 3.0 * n**3 / t
print r_n, r_n / (n * 1e-16)
```

- DGEMM reference implementation

```
import numarray, time
import numarray.random_array as naRA
n = 1000
a = naRA.random([n, n])
b = naRA.random([n, n])
c = naRA.random([n, n])
alpha = a[n/2, 0]
beta = b[n/2, 0]
t = -time.time()
c = beta * c + alpha * numarray.dot(a, b)
t += time.time()
print t, 2e-9 * n**3 / t
```

- STREAM reference implementation

```
import numarray, time
import numarray.random_array as naRA
import numarray.linear_algebra as naLA
m = 1000
a = naRA.random([m, 1])
alpha = naRA.random([1, 1])[0]
Copy, Scale = "Copy", "Scale"
Add, Triad = "Add", "Triad"
td = {}

td[Copy] = -time.time()
c = a[:]
td[Copy] += time.time()
td[Scale] = -time.time()
b = alpha * c
td[Scale] += time.time()
td[Add] = -time.time()
c = a * b
td[Add] += time.time()
td[Triad] = -time.time()
a = b + alpha * c
td[Triad] += time.time()
for op in (Copy, Scale, Add, Triad):
    t = td[op]
    s = op[0] in ("C", "S") and 2 or 3
    print op, t, 8.0e-9 * s * m / t
```

- PTRANS reference implementation

```
import numarray, time
import numarray.random_array as naRA
import numarray.linear_algebra as naLA
n = 1000
a = naRA.random([n, n])
b = naRA.random([n, n])
t = -time.time()
a = numarray.transpose(a)+b
```

```
t += time.time()
print t, 8e-9 * n**2 / t
```

- RandomAccess reference implementation

```
from time import time
from numarray import *
m = 1024
table = zeros([m], UInt64)
ran = zeros([128], UInt64)
mupdate = 4 * m
POLY, PERIOD = 7, 1317624576693539401L

def ridx(arr, i, tmp):
 tmp[0:1] = arr[i:i+1]
 if tmp.astype(Int64)[0] < 0:
 tmp <<= 1
 tmp ^= POLY
 else:
 tmp <<= 1
def starts(n):
 n = array([n], Int64)
 m2 = zeros([64], UInt64)

 while n[0] < 0:      n += PERIOD
 while n[0] > PERIOD: n -= PERIOD
 if n[0] == 0: return 1

 temp = array([1], UInt64)
 ival = array([0], UInt64)
 for i in range(64):
 m2[i] = temp[0]
 ridx(temp, 0, ival)
 ridx(temp, 0, ival)
    for i in range(62, -1, -1):
        if ((n>>i) & 1)[0]: break

    ran = array([2], UInt64)
```

```
    while (i > 0):
        temp[0] = 0
        for j in range(64):
            if ((ran>>j) & 1)[0]:
                temp ^= m2[j:j+1]
        ran[0] = temp[0]
        i -= 1
        if ((n>>i) & 1)[0]:
            ridx(ran, 0, ival)
    return ran[0]

ival = array([0], UInt64)

t = -time()
for i in range(m): table[i] = i
for j in range(128):
    ran[j] = starts(mupdate / 128 * j)
for i in range(mupdate / 128):
    for j in range(128):
        ridx(ran, j, ival)
        table[ran[j] & (m - 1)] ^= ran[j]
t += time()

temp = array([1], UInt64)
for i in range(mupdate):
    ridx(temp, 0, ival)
    table[temp & (m - 1)] ^= temp

temp = 0
for i in range(m):
    if table[i] != i: temp += 1

print t, 1e-9 * mupdate / t, 100.0*temp/m
```

- FFT reference implementation

```
import numarray, numarray.fft, time, math
import numarray.random_array as naRA
```

```
m = 1024
re = naRA.random([m, 1])
im = naRA.random([m, 1])
a = re + 1.0j * im

t = -time.time()
b = numarray.fft.fft(a)
t += time.time()

r = a - numarray.fft.inverse_fft(b)
r_n = numarray.maximum.reduce(abs(r))

Gflop = 5e-9 * m * math.log(m) / math.log(2)

print t, Gflop / t, r_n
```

## Acronyms

Explanation of acronyms used in this report:

1. CPU – Central Processing Unit
2. CSCS – Swiss Center for Scientific Computing
3. CVS – Concurrent Versions System
4. DARPA – The Defense Advanced Research Projects Agency
5. DFT – Discrete Fourier Transform
6. DGEMM – Double-precision General Matrix-Matrix multiply
7. FFT – Fast Fourier Transform
8. FFTE – an FFT code written in Fortran
9. FFTW – an FFT code written in C
10. HPCC – High Performance Computing Challenge Benchmark Suite
11. HPCS – High Productivity Computing Systems
12. HPL – High Performance Linpack benchmark
13. Linpack – LINear PACKage: a set of Fortran subroutines for numerical linear algebra; also a benchmark based on one of the Linpack subroutines
14. MPI – Message Passing Interface
15. NPB – NAS Parallel Benchmarks

16. PARKBENCH – PARallel Kernels and BENchmarks
17. PTRANS – Parallel matrix TRANSpose
18. PVM – Parallel Virtual Machine
19. RandomAccess – formerly known as GUPS benchmark
20. SLOC – Source Line of Code
21. STREAM – sustainable memory bandwidth test